VSCOdeGirls -- Tanzim Elahi, Jesse Hall, Ray Lee, Vishwaa Sofat
SoftDev1 pd1
P01 -- Task the First: The Plan
2019-11-15
Version 0

TERMS OF AGREEMENT (TOA):

1. Flask will serve as your web microframework.
2. Multiple supporting Python3 files will be used as necessary.
3. Bootstrap or Foundation will be used as your front-end framework.
4. You will provide your own customized CSS where appropriate/necessary.
5. You will make meaningful use (sum > parts) of at least three (3) REST APIs, chosen from Ye Olde SoftDev API KB.

FRAMEWORKS:

- HTML/CSS
- BootStrap
- Flask/Python3
- SQLite

APIs (API name/Pain factor):

- OpenSky Network API/0
- Google Maps Embedded/2
- IPLocation/0
- WebGL Earth API / 1 (has JavaScript)

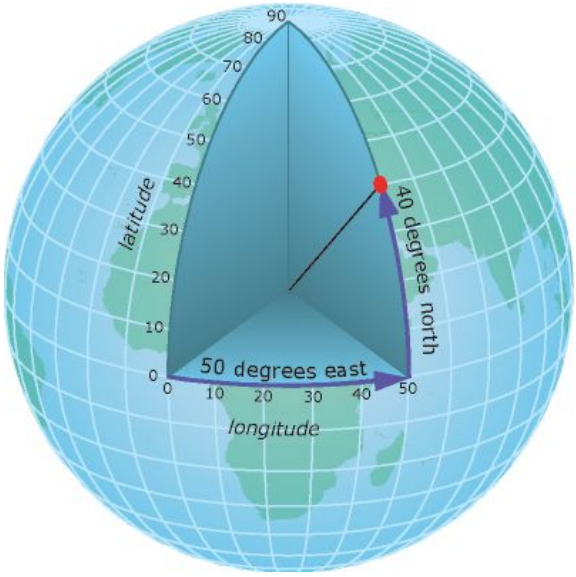
Minimum Viable Product (MVP)—TOA:number(s) in parentheses corresponds with the number in the terms of agreement above that it fulfills

- Create a flask app (TOA:1) that queries the REST APIs listed above to display nearby planes around a user's IP location or a custom set location (TOA:5).
- BootStrap and custom CSS used to liven up the page and make it look visually appealing (TOA:3,4)
- Use the Google custom search API to scrape for images related to the plane model of a certain nearby flight (TOA:5)
- SQLite will be used as a cache to store previously scraped images of a certain plane model in order to be more efficient and not query API as much (TOA:2, using python3 files to construct the database)
- updateLog.py is used to automate the log updating process (credits to Junhee Lee, who is the author of the script; TOA:2)

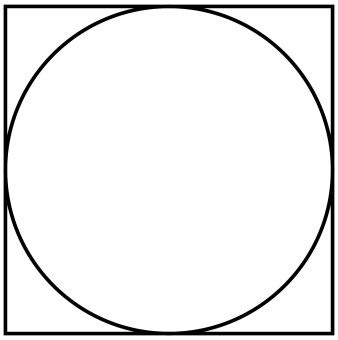
LIST OF COMPONENTS

- Accounts

    - *Create Account* → User will be able to go to a page where you can create an account. It will have you insert user information including the username and password. It will only be valid if the username does not already exist. Once created, the user's profile will be added to the accounts table of the SQLite database.
    - *Login* → The default route is a login page where it will ask for the username and password. This component will need access to the database to authenticate the user.
    - *Logout* → This will log the user out of their account and go back to the site's login page.
- Navigation Between Pages
    - *Menu* → At the top of the page, there will be a navbar with possible pages the user can visit or things the user can do.
        - Help Page
        - Login/Logout
        - Home Page
        - Fixed top navbar
        - Navbar buttons lead to different routes → displaying nearby planes around a custom address
- Search bar ("Search for nearby flights")
- Home screen - displays nearest planes within a certain user-defined radius around the approximate IP location of the user OR a custom set location
    - Dropdown menu → choose between pivoting radius around IP location or custom set location
    - Radius in miles (min: 10, max: 50; user-defined) → text box where user enters an integer that represents radius in miles
    - "Search for nearby flights" button → displays a list of nearby flights upon click, under a map that shows the location of the user as a generated image of a map
        - Each of the nearby planes is displayed as a text box (using BootStrap) below the map of the homescreen after the user enters a radius and presses "Search for nearby flights"
        - Each text box has a link to a route/generated page that displays specific information about the flight, plane model, model image, and other miscellaneous information
        - If the model has never been searched before, its image can be retrieved using Google custom search API. Otherwise, its image will be retrieved from cached data (image URLs) from the SQLite database.
    - Since the OpenSky API only takes in maximum and minimum latitude and longitude as parameters to define a rectangular area to search planes in, we must define this rectangular area into a square that also fits with the user-defined radius. After we have successfully queried the planes in the square area, we remove the planes whose distance from the user is greater than the radius.



- Ex:
- Range of latitude: [0; 90] degrees
- Range of longitude: [0; 180] degrees
- To get maximum and minimum latitude: user's latitude ± radius * (1 degree / 68.703 miles)
- For maximum and minimum longitude: user's longitude ± radius * (1 degree / 68.703miles)



- We must consider the edge case around the poles since the user-defined radius when the user is near the poles covers the latitude from their location up to the pole and over to the other side. Thus, we limit the upper bound of the latitude when the user-defined radius overflows near the north pole and limit the lower bound of the latitude when the user-defined radius overflows near the south pole.
  - 1 degree = 68.703 miles
  - At the north or south pole overflow, when the maximum or minimum latitude is >90 (respectively), fix the bounds to 90.
  - Longitude does not really have much complexities because when you reach 0 or 180, the bounds of longitude go back to 180 and 0, respectively.

- Once the search button is pressed, the user will be led to another page where there will be an interactable map (just a graph that overlays a map image

background), and the plot points that indicate nearby planes around the user. These plot points can be hovered over and clicked to lead to another route that shows more about that flight: current latitude and longitude, altitude, time of flight, etc.

TASKS

- Tanzim
    - Using MapQuest to measure distances between user and nearby planes
    - Help Ray/Jesse with latitude and longitude calculation Python methods
- Jesse
    - Using OpenSky to query area for nearby planes
- Ray (Project Manager)
    - Python skeleton
    - HTML skeleton (base.html and Jinja stuff)
    - BootStrap and custom CSS design
    - Help Jesse with OpenSky API
- Vishwaa
    - Using IPLocation to locate approximate location of user
    - Do HTML/CSS of the routes

DATABASE LAYOUT DIAGRAM

accounts table

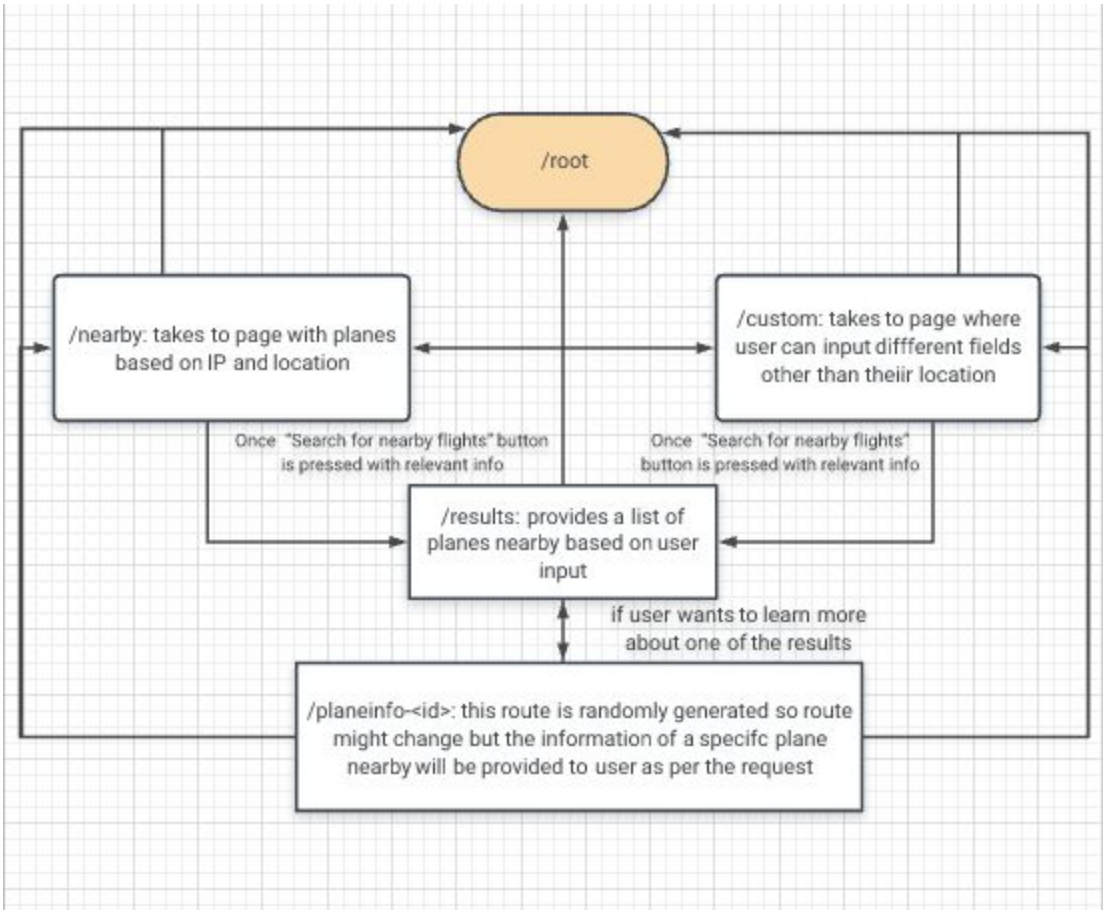
Username (TEXT)	*Password (TEXT)*
“pollsmor”	“AadUYbrfV5mLKXp7Dp93pcUUMwkyyE”
“xXx_yolo420gamer69_xXx”	“1234”
“iLoveFortnite”	“vbucksaregreat”
“...”	“...”

Some notes:

- If we have time, we should implement some password restrictions. Maybe:
    - At least 8 characters long
    - Must contain both numbers and letters

COMPONENT MAP

SITE MAP:


/root
/nearby: takes to page with planes based on IP and location
/custom: takes to page where user can input diffferent fields other than theiir location
Once "Search for nearby flights" button is pressed with relevant info
Once "Search for nearby flights" button is pressed with relevant info
/results: provides a list of planes nearby based on user input
if user wants to learn more about one of the results
/planeinfo-<id>: this route is randomly generated so route might change but the information of a specifc plane nearby will be provided to user as per the request